# **Lớp: IT003.O21**

**MSSV: 23521588**

**Họ và tên: Trần Thị Thùy Tiên**

**Bài tập Thuật toán tìm kiếm**

1. **Linear Search:**
2. Ý tưởng:

* Duyệt qua tất cả các phần tử của mảng, nếu xuất hiện 1 giá trị bằng với giá trị x đang tìm kiếm -> thì trả về true. Ngược lại, không tìm thấy -> trả về false.

1. Biểu diễn thuật toán:

Bool linearSearch(int \*a, int n, int x)

{

for (int i = 0; i < n; i++)

if (a[i] == x) return true;

return false;

}

1. Minh họa thuật toán:

- Dãy a : 54, 26, 93, 17, 77, 31, 44, 55, 20

- Tìm x1 = 54

+ Lần 1: i = 0 -> a[0] = 54 -> so sánh với x1 ( x1 = a[0] = 54) -> hàm trả về true, và kết thúc hàm

- Tìm x2 = 6

+ Lần 1: i = 0 -> a[0] = 54 != x2 -> thực hiện lại vòng lặp

+ Lần 2: i = 1 -> a[1] = 26 != x2 -> thực hiện lại vòng lặp

+ Lần 3: i = 2 -> a[2] = 93 != x2 -> thực hiện lại vòng lặp

+ Lần 4: i = 3 -> a[3] = 17 != x2 -> thực hiện lại vòng lặp

+ Lần 5: i = 4 -> a[4] = 77 != x2 -> thực hiện lại vòng lặp

+ Lần 6: i = 5 -> a[5] = 31 != x2 -> thực hiện lại vòng lặp

+ Lần 7: i = 6 -> a[6] = 44 != x2 -> thực hiện lại vòng lặp

+ Lần 8: i = 7 -> a[7] = 55 != x2 -> thực hiện lại vòng lặp

+ Lần 9: i = 8 -> a[8] = 20 != x2 -> thực hiện lại vòng lặp

+ Lần 10: i = 9 = n -> không thỏa mãn điều kiện -> thoát chương trình và trả về false.

* + Tìm x3 = 20:

+ Lần 1: i = 0 -> a[0] = 54 != x2 -> thực hiện lại vòng lặp

+ Lần 2: i = 1 -> a[1] = 26 != x3 -> thực hiện lại vòng lặp

+ Lần 3: i = 2 -> a[2] = 93 != x3 -> thực hiện lại vòng lặp

+ Lần 4: i = 3 -> a[3] = 17 != x3 -> thực hiện lại vòng lặp

+ Lần 5: i = 4 -> a[4] = 77 != x3 -> thực hiện lại vòng lặp

+ Lần 6: i = 5 -> a[5] = 31 != x3 -> thực hiện lại vòng lặp

+ Lần 7: i = 6 -> a[6] = 44 != x3 -> thực hiện lại vòng lặp

+ Lần 8: i = 7 -> a[7] = 55 != x3 -> thực hiện lại vòng lặp

+ Lần 9: i = 8 -> a[8] = 20 = x3 -> trả về true và thoát hàm

1. Cài đặt:

#include<iostream>

using namespace std;

int findIdX (int a[], int n, int x) {

for (int i = 0; i < n; i++)

if (a[i] == x) return true;

return false;

}

int main () {

int n, x;

cin >> n >> x;

int \*a;

a = new int[n];

for (int i = 0; i < n; i++)

cin >> a[i];

cout << findIdX(a, n, x);

}

1. Độ phức tạp:

O(n)

1. So sánh:

+ Nếu cho dãy tăng dần thì thuật toán Linear Search sẽ có độ phức tạp lớn thuật toán Binary Search.

+ Còn đối dãy không theo trực tự thì thuật toán tối ưu hơn với bài toán sử dụng thuật Binary Search.

1. **Binary Search:**
2. Ý tưởng:

- Với dãy tăng, ta tìm phần tử chính giữa mảng (a[m]) và so sánh với giá trị x

+ Nếu bằng x thì ta trả về true và kết thúc hàm

+ Nếu x > a[m] -> ta chỉ cần tìm trong đoạn bên phải phần tử a[m]. Ngược lại, ta chỉ cần tìm trong đoạn bên trái phần tử a[m].

-> lặp lại cho đến khi không có đoạn nào để tìm kiếm -> ta trả về false và kết thúc hàm.

1. Biểu diễn thuật toán

Bool binarySearch(int \*a, int n, int x)

{

int l = 0, r = n – 1;

While(l <= r)

{

Int mid = (l + r) / 2;

If (a[mid] == x) return true;

If (a[m] < x) l = mid + 1;

Else r = mid – 1;

}

Return false;

}

1. Minh họa thuật toán:

- Dãy a : 54, 26, 93, 17, 77, 31, 44, 55, 20

-> Sắp xếp lại mảng a : 17 20 26 31 44 54 55 77 93

- Tìm x1 = 54

+ Lần 1: l = 0, r = 8 -> mid = 4 -> a[4] = 44 < x1 -> l = 5

+ Lần 2: l = 5, r = 8 -> mid = 6 -> a[6] = 55 > x1 -> r = 5

+ Lần 3: l = 5, r = 5 -> mid = 5 -> a[5] = 54 = x1 -> trả về true và kết thúc hàm

-Tìm x2 = 6

+ Lần 1: l = 0, r = 8 -> mid = 4 -> a[4] = 44 > x3 -> r = 3

+ Lần 2: l = 0, r = 3 -> mid = 1 -> a[1] = 20 > x3 -> r = 0

+ Lần 3: l = 0, r = 0 -> mid = 0 -> a[0] = 17 > x3 -> r = -1

+ Lần 4: l = 0, r = -1 -> không thỏa mãn điều kiện -> thoát vòng lặp và trả về false, kết thúc hàm.

-Tìm x3 = 20

+ Lần 1: l = 0, r = 8 -> mid = 4 -> a[4] = 44 > x3 -> r = 3

+ Lần 2: l = 0, r = 3 -> mid = 1 -> a[1] = 20 = x3 -> trả về true và kết thúc hàm

1. Cài đặt:

#include<iostream>

using namespace std;

void input(int \*a, int &n, int &x) {

cin >> n >> x;

a = new int[n];

for (int i = 0; i < n; i++)

cin >> a[i];

}

bool binarySearch(int \*a, int l, int r, int x) {

if (l > r) return false;

int mid = (l + r) / 2;

if (a[mid] == x) return true;

if (a[mid] < x) return binarySearch(a, mid + 1, r, x);

return binarySearch(a, l, mid-1, x);

}

int main () {

int n, int x;

int \*a;

input(a, n, x);

// Thực hiện sắp xếp mảng a

cout << binarySearch(a, n, x);

}

1. Độ phức tạp:

- Độ phức tạp thuật toán binary search : O(logN)

- Độ phức tạp bài toán phụ thuộc và độ phức tạp của thuật toán sắp xếp

1. So sánh:

- Với bài toán, cho 1 dãy tăng dần và tìm kiếm giá trị x -> thì sử dụng thuật toán này sẽ tối ưu hơn so với thuật toán Linear Search

- Còn với bài toán, cho 1 dãy không có trực tự và tìm kiếm giá trị x -> thì khi sử dụng thuật toán Binary Search cho bài toán sẽ không tối ưu vì nó còn phụ thuộc và độ phức tạp của các thuật toán Sort.